Niveau(x) d'enseignement: 0 Numéro EDUQ : 8570295
Centre d'intérêt: 640

Dalois, Maurice.

Les langages de programmation : le langage APL / animateur [et élaboré par] Maurice Dalois ; [pour le] Service de la formation du personnel scolaire, Direction générale du développement pédagogique. -- [Québec] : La Direction, 1985.
41 p. ; 28 cm. -- (Guide de formation / Programme de formation lourde en micro-informatique appliquée à l'éducation ; 6.3)

Titre de la couv.

Descripteur(s): *Informatique *Programmation *Application pédagogique de l'ordinateur *Guide de l'enseignant

Identificateur(s) de sujet: *Langage de programmation *APL (Langage de programmation)

Le guide de formation, produit dans le cadre du "Programme de formation lourde en micro-informatique appliquée à l'éducation", fait partie d'une série comprenant une vingtaine de documents. Ces guides ont été produits pour les agents multiplicateurs comme documents d'accompagnement aux différents cours donnés dans le cadre de la formation lourde. L'objectif du document est de familiariser les participants au langage de programmation APL. Trois chapitres et une annexe composent le document: 1. APL: un super calculateur; 2. à la découverte des primitives; 3. la programmation avec APL; annexe: l'éditeur plein-écran d'APL * PLUS/PC.

Type de document: d

EDUQ 8570295

SERVICE DE LA FORMATION DU PERSONNEL SCOLAIRE

PROGRAMME DE FORMATION LOURDE EN MICRO-INFORMATIQUE APPLIQUEE A L'EDUCATION

GUIDE DE FORMATION

VOLET

LES LANGAGES DE PROGRAMMATION

SECTION

LE LANGAGE APL

FEVRIER 1985

ANIMATEUR/TRICE

MAURICE DALOIS

CO-ANIMATEUR/TRICE

6.3

Gouvernement du Québec
Ministère de l'Education
Direction générale
du développement pédagogique

AVANT-PROPOS

Le présent guide de formation a été produit dans le cadre du Programme de formation lourde en micro-informatique appliquée à l'éducation. Ce programme de formation a été développé par le Service de la formation du personnel scolaire au ministère de l'Education. Le programme est d'une durée de 30 jours et comprend huit volets de formation. Le lecteur trouvera les explications relatives à ces différents volets dans un document intitulé "Présentation du programme de formation lourde".

Ce guide de formation fait partie d'une série comprenant une vingtaine de documents. La coordination de la publication de cette série de guides a été assurée par Normand Gagné. Ces guides ont été produits pour les agents multiplicateurs comme documents d'accompagnement aux différents cours donnés dans le cadre de la formation lourde.

Ces documents complètent aussi les informations contenues dans les guides de sensibilisation et de formation légère prévus pour aider les agentes et agents multiplicateurs dans leur future tâche de formateur dans leur propre milieu scolaire. Tous ces documents peuvent évidemment être reproduits librement à ces fins.

Nous espérons que ces guides de formation sauront aider ceux et celles que les applications pédagogiques de l'ordinateur intéressent.

Jean-Guy Godbout
Directeur général adjoint

Ce document a été élaboré

par

MAURICE DALOIS

TABLE DES MATIERES

Page

INTRODUCTION 1

CHAPITRE 1 - APL: UN SUPER CALCULATEUR! 3

1.1 APL est plus qu'une calculatrice ordinaire 4
1.2 Enchaînons des opérations 6
1.3 Manipulons maintenant du texte 7

CHAPITRE 2 - A LA DECOUVERTE DES PRIMITIVES 9

2.1 Le générateur d'entiers 9
2.2 Nombres aléatoires 10
2.3 Ensembles de nombres 11
2.4 Manipulons des notes d'élèves 12
2.5 Manipulons un peu de texte 19

CHAPITRE 3 - LA PROGRAMMATION AVEC APL 21

3.1 Premières fonctions 22
3.2 Fonctions avec 2 arguments 27
3.3 Fonction avec branchements 29
3.4 Commandes de systèmes élémentaires 34

ANNEXE - L'éditeur plein-écran D'APL * PLUS / PC 35

INTRODUCTION

APL*PLUS/PC de la Compagnie STSC (Scientific Time Sharing Corporation) est une version d'APL spécialement adaptée au micro ordinateur IBM/PC. Cette version pour micro diffère passablement des versions d'APL déjà développées pour les gros ordinateurs en ce qui concerne les facilités d'édition et de gestion de l'écran.

En plus du noyau standard d'APL cette version offre des fonctions graphiques, un éditeur plein écran permettant l'accès aux caractères français, un système de fichier APL très "amical" ainsi que l'accès aux fichiers du DOS, l'accès à toutes les fonctions du DOS sous APL, et enfin citons des fonctions de communication permettant la communication avec d'autres ordinateurs (ou des périphériques divers) à l'intérieur même d'APL.

La version 3.1 d'APL*PLUS/PC que nous utiliserons exige un minimum de 192Kb de mémoire centrale et une unité de disquette.

EN MARCHE ...

- Insérez la disquette **système** APL dans l'unité de gauche
- Démarrez l'ordinateur et l'écran
- Répondez si vous le désirez à la demande de DATE et HEURE
- Le chargement de l'interprète APL se fait en une dizaine de secondes et vous obtenez, après l'identification de la compagnie, le message **CLEAR WS Date et Heure ...**

Le système est alors prêt à recevoir des commandes. On dira qu'il est en "**mode d'exécution immédiate**". On peut s'en servir au début comme d'un super calculateur (voir chapitre 1).

le clavier APL

ATTENTION, le clavier APL est différent du clavier standard ! c'est un obstacle qu'il faut à tout prix surmonter. l'effort en vaut la peine !!

Vous serez sûrement incommodés au départ pour trouver les symboles APL. Sur le clavier APL, les lettres et les chiffres sont à la même place que sur le clavier standard, mais, malheureusement, les caractères spéciaux comme ? : ; ( ) + - < > sont sur des touches différentes. Les raisons de ces différences entre le clavier APL et le clavier standard sont historiques ... Nous n'y pouvons malheureusement rien!

En revanche, le **clavier APL** est un clavier enrichi: on y trouve en particulier les vrais symboles de la multiplication × et de la division ÷ qui font cruellement défaut sur le clavier ASCII standard, ainsi que les symboles ≠, ≤ et ≥. Et enfin on trouve sur ce clavier APL tous les nouveaux symboles du langage APL.

Notez que le clavier standard n'est pas pour autant éliminé: en appuyant sur la touche NUMLOCK (auto-collant Text/APL) on retrouve le clavier standard.

A défaut de touches gravées avec les symboles APL nous utilisons des **auto-collants** placés sur l'avant des touches. Lorsque nous sommes en "clavier APL" il faut lire la signification sur l'avant des touches. On remarque que les symboles sont de 3 couleurs:

- Les caractères **verts** qui sont obtenus en tenant enfoncée la touche en vert (SHIFT) et en appuyant sur la touche désirée.
- Les caractères **rouges** qui sont obtenus en tenant enfoncée la touche en rouge (ALT) et en appuyant sur la touche désirée.
- Les caractères **noirs** obtenus par simple appui sur la touche désirée.

Commencez par trouver les symboles + − × ÷ et vous pourrez passer au chapitre 1.

Les **caractères français** enfin peuvent être obtenus directement au clavier par superposition (procédure quelque peu lourde). Par exemple pour le **é** on fait: e , overstrike , / "overstrike" est la touche "0 Ins" du clavier numérique. Une autre façon consiste à programmer les clés de fonction ce qui est très facile sous APL, ou encore il est possible de redéfinir certaines touches du clavier.

CHAPITRE 1

APL: UN SUPER CALCULATEUR!

APL exécute immédiatement les opérations entrées à partir du clavier:

```
      3+12      ←+ expression tapée et suivie de ENTER
15
```

15 est la réponse de l'ordinateur

On remarque que le curseur se trouve normalememt en attente d'une commande en colonne 7, tandis que la réponse s'écrit à partir de la colonne 1.

```
      3+12÷7.5
22.5
      4×150     APL possède le symbole habituel de la multiplication
600
      12 - 18
¯6
```

on remarque que le signe d'un nombre est distinct de l'opération SOUSTRACTION

```
      ¯7 + ¯3
¯10
```

le signe d'un nombre est sur la même touche le nombre 2 (SHIFT 2)

```
      18÷3
6
      4÷3
1.333333333
      4÷¯3
¯1.333333333
      3*2          le symbole  *  élève 3 à la puissance 2
9
      2*10
1024
      2*0.5        racine carrée de 2
1.414213562
```

1.1 APL EST PLUS QU'UNE CALCULATRICE ORDINAIRE

Pour calculer la taxe de 8% sur des montants successifs de 55, 76.5 et 2.25 $, on peut faire successivement les 3 opérations suivantes:

```
      .08×55
4.4
      .08×76.5
6.12
      .08×2.25
0.18
```

Mais on peut aussi faire une seule opération **globale** sur l'ensemble des 3 montants:

```
      .08 × 55  76.5  2.25
4.4 6.12 0.18
```

De même si l'on a: 12 articles à 55$
8 articles à 76.5$
22 articles à 2.25$

Au lieu de faire comme avec une calculatrice ordinaire:

```
      12×55
660
      8×76.5
612
      22×2.25
49.5
```

on peut faire:

```
      12 8 22 × 55 76.5 2.25
660 612 49.5
```

on voit que l'opération se fait terme à terme comme si l'on avait fait: (12×55),(8×76.5),(22×2.25)

Enchaînons pour calculer la taxe sur chaque article vendu en plusieurs exemplaires:

```
      .08 × 12 8 22 × 55 76.5 2.25
52.8 48.96 3.96
```

Donnons des noms significatifs à ces valeurs

```
      PRIX←55 76.5 2.25
      QUANTITE←12 8 22
```

Les mots PRIX et QUANTITE sont appelés des **variables** (Ils peuvent représenter diverses valeurs), et la flèche ← indique de façon assez imagée que les valeurs à droite sont attribuées au mot. Alors les calculs précédents sont obtenus de façon naturelle en tapant:

```
      PRIX × QUANTITE
660 612 49.5
```

```
      .08×PRIX×QUANTITE
52.8 48.96 3.96
```

Enfin, la somme des taxes peut être obtenue directement par:

```
      +/.08×PRIX×QUANTITE
105.72
```

+/ fait la somme de tous les nombres qui se trouvent à droite

Nous venons d'aborder une des caractéristiques importantes d'APL:
Les opérations GLOBALES sur des ensembles de données.

Pour voir le contenu d'une variable on tape son nom:

```
      PRIX
35 76.5 2.25
      QUANTITE
12 6 22

      TAXE← 0.08 × PRIX × QUANTITE
```

NB: Quand il y a affectation (←) dans une variable, la valeur ne s'imprime pas: elle est "envoyée" dans la variable. Pour avoir cette valeur il suffit, comme on l'a déjà vu, de taper son nom.

```
      TAXE
52.8 48.96 3.96
```

Si,par contre, on ne fait pas d'affectation, alors le résultat de l'exécution s'imprime:

```
      0.08 × PRIX × QUANTITE
52.8 48.96 3.96
```

1.2 ENCHAINONS DES OPERATIONS:

```
      4 × 3 + 2
20
```

Le résultat peut surprendre! Voici:
APL fait d'abord 3+2 qui donne 5,
puis 4×5 qui font 20.

Règle importante: APL exécute chaque ligne en commençant par l'opération qui est la plus à droite, sans aucune règle de priorité entre les opérations. Seules des parenthèses modifient l'ordre d'exécution de droite à gauche

```
      (4×3)+2    l'expression entre parenthèse s'exécute en priorité
14
```

notez que l'on peut mettre des parenthèses même si elles sont inutiles:

```
      4×(3+2)
20
```

NOTE: La règle d'opération de **droite à gauche** trouve sa justification dans la notation des fonctions en mathématique et elle simplifie grandement les règles traditionnelles (et arbitraires) de priorité que l'on rencontre dans la plupart des autres langages.
D'abord surprenante, à cause de l'habitude que nous avons d'exécuter une multiplication avant une addition dans une formule mathématique, cette règle s'avère être un choix judicieux et simplificateur dès que l'on a un grand nombre d'opérations primitives dans le langage.

Voici donc quelques exemples additionnels:

```
      15-9-3        ←→  c'est comme  15-(9-3)
9
      3×4-5×2       ←→  l'expression s'exécute comme  3×(4-(5×2))
¯18
      (3×4)-5×2     ←→  est équivalent à  (3×4)-(5×2)
2
```

Il en est naturellement de même avec des variables:

```
      A←12
      B←9
      C←4
      A×B+C
156
```

Si l'on veut faire la multiplication avant l'addition on met des parenthèses:

```
      (A×B)+C
112
```

ou encore on inverse l'ordre des opérations:

```
      C+A×B
112
```

1.3 MANIPULONS MAINTENANT DU TEXTE:

Les données en caractère doivent être placées entre apostrophes pour ne pas être confondues avec des noms de variable lorsque l'on se trouve en mode d'exécution immédiate (calculateur).
Par exemple:

```
      ADRESSE←'1234 DE LA MONTAGNE, QUEBEC'
```

en tapant le mot ADRESSE on obtient le contenu:

```
      ADRESSE
1234 DE LA MONTAGNE, QUEBEC

      NOM←'ARTHUR'
      'BONJOUR ',NOM,' COMMENT VAS-TU?'
BONJOUR ARTHUR COMMENT VAS-TU?
```

La virgule (,) permet de mettre consécutivement (concaténer) des données en caractère (des **chaînes de caractères**).

Si le texte contient lui-même des apostrophes il faut doubler chaque apostrophe. Par exemple on tapera 'JE M''EN VAIS' pour obtenir le texte JE M'EN VAIS. Voici:

```
      T←'JE M''EN VAIS'
      T
JE M'EN VAIS
```

Ne pas confondre le nom d'une variable et sa valeur (son contenu). Par exemple, on peut avoir la variable BONBON dont la valeur est 'BONBON' !

```
      BONBON←'BONBON'
      BONBON
BONBON
```

On ne devra pas confondre non plus, une valeur numérique (un nombre), et une valeur caractère (ce nombre mis entre apostrophes):

```
      A←1983      on peut faire des calculs avec A
      B←'1983'    on ne peut pas faire de calculs avec B qui
                  est de type caractère.
```

Le clavier du IBM/PC permet d'entrer également avec APL du texte en minuscules ainsi que les voyelles accentuées. Il faut se placer en "mode Text" en appuyant sur la touche NUMLOCK identifiée par l'auto-collant "Text/APL". Les lettres accentuées peuvent être obtenues par superposition ou par programmation des clés de fonction (voir l'introduction sur le clavier).

```
      TEXTE←'Maître corbeau sur un arbre perché ...'
      TEXTE
Maître corbeau sur un arbre perché ...
```

Après entrée du texte on revient au clavier APL en appuyant à nouveau sur la touche "Text/APL".

CHAPITRE 2

A LA DECOUVERTE DES PRIMITIVES

Nous aborderons les principales fonctions primitives d'APL à travers des exercices pratiques simples. On peut trouver une définition formelle de ces primitives dans la documentation qui accompagne le logiciel. Le lecteur pourra s'y reporter pour approfondir chaque fonction primitive.

Il n'est pas nécessaire de chercher à comprendre en détail les expressions APL qui seront proposées dans ce chapitre. L'objectif est de saisir "l'esprit plutôt que la lettre". C'est un peut comme l'apprentissage d'une nouvelle langue par une méthode globale: on plonge d'abord dans le texte et ensuite on étudie plus attentivement la structure de la langue et la grammaire. lus la langue et la

2.1 -LE GENERATEUR D'ENTIERS ⍳ (IOTA)

```
      ⍳20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

la primitive ⍳ génère des entiers à partir de 1 jusqu'à la valeur entière donnée à droite.

```
      N←25
      ⍳N
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Pour générer les 20 premiers multiples de 2:

```
      2×⍳20
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

Pour générer les 50 premiers nombres impairs:

```
      (2×⍳50)-1
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55
57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Pour obtenir tous les entiers entre 100 et 120 inclus:

```
      99+⍳21
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
 120
```

Pour compter en dixièmes entre 3 et 5

```
      3+(⍳20)÷10
3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5
```

2.2 - NOMBRES ALEATOIRES

- LA LOTTO 6/49 :il s'agit de produire 6 nombres aléatoires sans répétition entre 1 et 49. Voici un premier tirage:

```
      6?49
14 36 21 9 23 48
```

et voici un 2ième tirage:

```
      6?49
19 12 43 27 42 37
```

- **Lancement de dés.** C'est un tirage aléatoire avec répétition. On utilise alors le "?" avec un argument à droite. On simule le lancement de 3 dés par:

```
      ?6 6 6
2 2 5
encore:
      ?6 6 6
6 4 3
encore:
      ?6 6 6
4 4 2
```

Lançons 20 fois de suite un dé (ou 20 dés en même temps!):

```
      20⍴6     répète 20 fois le nombre 6:
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
      ESSAI←?20⍴6
      ESSAI
2 3 2 4 3 1 2 5 1 6 6 2 4 3 3 3 2 5 3 2
```

pour chacun des 6 on obtient un nombre aléatoire entre 1 et 6

Combien de fois a-t-on obtenu le chiffre 6?

```
      ESSAI=6
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
```

donne un 1 correspondant à chaque 6. Il suffit d'en faire la somme:

```
      +/ESSAI=6
2
```

- **Pour obtenir une suite aléatoire des entiers de 1 à 10:**

```
      10?10
4 7 10 8 6 3 9 1 5 2
```

Application: mélanger les lettres d'un mot

```
      MOT←'MONTREAL'
      ⍴MOT
8
      MOT[8?8]   8?8 donne les indices "mélangés"
EAMRLONT
```

2.3 - ENSEMBLES DE NOMBRES

- Créons 2 variables A et B de nombres entiers de 1 à 50

```
      A←10?50   10 nombres aléatoires sans répétition entre 1 et 50
      B←15?50   15 nombres aléatoires sans répétition entre 1 et 50
```

Voici les valeurs de A et B:

```
      A
27 15 5 2 37 3 7 22 43 50
      B
20 48 16 17 46 14 11 28 45 2 15 8 5 3 33
```

- Quels sont les éléments communs à A et B?

```
      A∈B
0 1 1 1 0 1 0 0 0 0
```

la primitive ∈ donne un vecteur de 1 et de 0 selon que chaque élément de A appartient ou non à B.

Pour extraire les éléments communs on dispose de la fonction primitive / qui **sélectionne** les éléments de A correspondant à un 1 et qui élimine les autres:

```
      0 1 1 1 0 1 0 0 0 0 / A
15 5 2 3
```

L'intersection de A et B est donc obtenue par:

```
      (A∈B)/A
15 5 2 3
```

ou encore (c'est symétrique) par:

```
      (B∈A)/B
2 15 5 3
```

- Combien y a-t-il d'éléments de A dans B ?

```
      +/A∈B    il suffit de compter le nombre de 1
4
```

ou bien:

```
      ⍴(A∈B)/A   c'est le nombre d'éléments de l'intersection
4
```

- Quels sont les éléments qui sont dans A et pas dans B?

il suffit d'inverser les 0 et 1 avec la primitive ~ (négation)

```
      ~A∈B
1 0 0 0 1 0 1 1 1 1
```

donc, le résultat est:

```
      (~A∈B)/A
27 37 7 22 43 50
```

Si on a un troisième vecteur C:

```
      C←2 3 5 7
```

Est-ce que C est dans A ou dans B?

```
      C∈A
1 1 1 1
      ∧/C∈A   donne 1 si tous les éléments sont des 1 (et logique)
1
      ∧/C∈B   C n'est pas dans B
0
```

2.4- MANIPULONS DES NOTES D'ELEVES

Soit un groupe d'élèves (pour la circonstance un groupe de 10 élèves seulement) dont les notes sur 100 sont les suivantes:

```
      NOTES←78 90 68 95 45 100 59 82 40 80
```

On dit que la variable NOTES est un **vecteur numérique.**

On peut extraire la note du 2ième élève en donnant son **indice** c'est à dire sa position, par:

```
      NOTES[2]
90
```

Et les notes des 2ième, 6ième et 3ième, dans cet ordre, par:

```
      NOTES[2 6 3]
90 100 68
```

Pour trouver la moyenne du groupe il faut d'abord faire la somme des notes... Voici:

```
      +/NOTES
737
```

+/ fait la somme des éléments d'un vecteur numérique

Le nombre de notes (nombre d'éléments du vecteur NOTES) est obtenu par:

```
      ⍴NOTES
10
```

⍴ donne la **dimension** (nombre d'éléments) d'un vecteur

La moyenne du groupe est donc:

```
      (+/NOTES)÷⍴NOTES
73.7
```

Deux élèves viennent s'ajouter à la classe et leurs notes sont 55 et 83. On ajoute ces notes au vecteur NOTES comme suit:

```
      NOTES←NOTES,55 83
      NOTES
78 90 68 95 45 100 59 82 40 80 55 83
      ⍴NOTES
12
```

Notez qu'il n'est pas nécessaire de réserver l'espace mémoire pour la variable NOTES dont la dimension peut augmenter à volonté dans les limites de l'espace mémoire disponible. Avec APL pas besoin d'énoncé DIMENSION !

La nouvelle moyenne est donc:

```
      (+/NOTES)÷⍴NOTES
72.91666667
```

Combien y a-t-il de notes > 60 ?. Voici:

```
      NOTES>60
1 1 1 1 0 1 0 1 0 1 0 1
```

On obtient un vecteur de 1 et 0 (vecteur binaire ou booléen), qui indique par un 1 les notes supérieures à 60. il suffit ensuite de faire +/ ...

```
      +/NOTES>60
8
```

il y a 8 notes supérieures à 60.

Quelle est la meilleure note?

```
      ⌈/NOTES      la primitive ⌈ est appelée "maximum".
100
```

Quelle est la note la plus faible?

```
      ⌊/NOTES      la primitive ⌊ est appelée "minimum".
40
```

Pour ordonner (trier) ces notes en ordre décroissant on dispose d'une fonction primitive ⍒ qui opère comme suit:

```
      I←⍒NOTES
      I
6 4 2 12 8 10 1 3 7 11 5 9
```

le résultat est le vecteur des **indices** dans l'ordre. C'est à dire: le premier élément 6 indique que la première note vient en 6ième position, puis vient la 4ième, la 2ième , etc...
Pour avoir les notes dans l'ordre décroissant on fait alors:

```
      NOTES[I]
100 95 90 83 82 80 78 68 59 55 45 40
```

Ce qui peut être obtenu directement par:

```
      NOTES[⍒NOTES]
100 95 90 83 82 80 78 68 59 55 45 40
```

ASSOCIONS LES NOMS DES ELEVES

Il existe plusieurs façons de lire les noms et de les placer dans une variable sous la forme d'un **tableau**, comme par exemple la variable NOMS suivante:

```
      NOMS
RIMBAULT        ARTHUR
DAUDET          ALPHONSE
ZOLA            EMILE
POQUELIN        JEAN-BAPTISTE
DESCARTES       RENE
LA FONTAINE     JEAN
BOILEAU         NICOLAS
LA BRUYERE      JEAN
PROUST          MARCEL
DIDEROT         DENIS
ROUSSEAU        JEAN-JACQUES
CORNEILLE       PIERRE
```

La variable NOMS est aussi appelée **matrice** en APL (ce mot **matrice** qui nous vient des mathématiques ne doit pas être pris au sens stict des mathématiques). On peut connaître la dimension du tableau (nombre de lignes et nombre de colonnes) avec la primitive ⍴:

```
      ⍴NOMS
12 29
```

la variable NOMS est un tableau de 12 lignes par 29 colonnes. Pour définir la variable NOMS nous suggérons d'utiliser l'éditeur d'APL (cet éditeur est propre à APL*PLUS/PC).

On appelle l'éditeur par la commande de système **)EDIT nnn** où **nnn** est le nom de la variable que l'on veut éditer.

Attention: la parenthèse ")" est celle d'APL - voir la définition du clavier. Toutes les commandes de système commencent par une parenthèse droite ")"

```
)EDIT NOMS
```

Après cette commande vous êtes sous le contrôle de l'éditeur:

- l'écran est vidé et le curseur se positionne en ligne[0]
- la ligne 25 de l'écran indique que le type de variable est par défaut V (pour vecteur). Il faut changer le type pour une matrice. Vous tapez **Ctrl A** puis vous répondez à la ligne 25 par M (pour Matrice).
- vous entrez les noms et prénoms avec l'alignement désiré
- vous pouvez remonter le curseur, corriger le texte à volonté...
- pour sortir de l'éditeur vous faites **Ctrl E**
- vous retrouvez alors l'écran initial.

(les commandes de l'éditeur sont données dans l'annexe B)

La variable NOMS contient les noms des élèves. On peut le vérifier en tapant: NOMS

Pour affichier les NOMS et les NOTES, il faut au préalable restructurer les notes en colonnes. Dans l'exemple nous avons 12 notes. On les restructure en un "tableau" (ou matrice) de 12 lignes par 1 colonne en faisant:

```
     12 ⍴NOTES
 78
 90
 68
 95
 45
100
 59
 82
 40
 80
 55
 83
```

Il faut enfin convertir les NOTES en caractère avec la primitive ⍕ avant de les placer (concaténer) à droite des NOMS avec la virgule. Voici:

```
     NOMS,⍕12 ⍴NOTES
RIMBAULT       ARTHUR          78
DAUDET         ALPHONSE        90
ZOLA           EMILE           68
POQUELIN       JEAN-BAPTISTE   95
DESCARTES      RENE            45
LA FONTAINE    JEAN           100
BOILEAU        NICOLAS         59
LA BRUYERE     JEAN            82
PROUST         MARCEL          40
DIDEROT        DENIS           80
ROUSSEAU       JEAN-JACQUES    55
CORNEILLE      PIERRE          83
```

La primitive ⍕ (format) a pour effet de convertir les NOTES (qui sont de type numérique) en type **caractère** afin de les "concaténer" avec les NOMS qui sont de type caractère. Si l'on omet la primitive ⍕ on otient le message "DOMMAIN ERROR" qui signifie l'impossibilité d'associer des données de type différent. En d'autres termes, le DOMAINE d'application de la fonction est incorrect. Essayons:

```
     NOMS,12 ⍴NOTES
DOMMAIN ERROR
     NOMS,12 ⍴NOTES
          ▲
```

le symbole ^ indique l'endroit où s'est produite l'erreur

TRAITONS UN TABLEAU DE NOTES

Supposons maintenant les notes de 4 examens pour ce groupe de 12 élèves. On peut former ce tableau directement avec la primitive ⍴ en donnant à gauche la dimension 12 4 (12 lignes par 4 colonnes).

```
      NOTES←12 4 ⍴ 80 78 45 95 97 85 46 60 65 100 88 75 66 60 70 75 78 0 100 96
 95 80 80 80 72 55 48 33 50 60 98 76 89 100 98 95 60 60 100 87 80 80 93 55 100
20 56 75
      NOTES
  80   78   45   95
  97   85   46   60
  65  100   88   75
  66   60   70   75
  78    0  100   96
  95   80   80   80
  72   55   48   33
  50   60   98   76
  89  100   98   95
  60   60  100   87
  80   80   93   55
 100   20   56   75
```

Pour extraire de ce tableau appelé encore **matrice**, on peut utiliser les **indices**. Par exemple la note du 7ième élève au 2ième examen est obtenue par:

```
      NOTES[7;2]   un ; sépare les indices de ligne et colonne
55
```

Pour avoir les 2ième et 3ième notes de ce 7ième élève:

```
      NOTES[7;2 3]
55 48
```

Pour avoir toutes les notes de cet élève:

```
      NOTES[7;]    l'absence d'indice de colonne signifie "tout"
72 55 48 33
```

Voici toutes les notes des 2 3 et 7ième élèves:

```
      NOTES[2 3 7;]
  97   85   46   60
  65  100   88   75
  72   55   48   33
```

Enfin voici les notes du 3ième examen pour toute la classe:

```
      NOTES[;3]    le résultat est un vecteur
45 46 88 70 100 80 48 98 98 100 93 56
```

Ces nouvelles NOTES peuvent être associées aux NOMS directement par:

```
      NOMS,⍪NOTES
RIMBAULT        ARTHUR          80  78  45  95
DAUDET          ALPHONSE        97  85  46  60
ZOLA            EMILE           65 100  88  75
POQUELIN        JEAN-BAPTISTE   66  60  70  75
DESCARTES       RENE            78   0 100  96
LA FONTAINE     JEAN            95  80  80  80
BOILEAU         NICOLAS         72  55  48  33
LA BRUYERE      JEAN            50  60  98  76
PROUST          MARCEL          89 100  98  95
DIDEROT         DENIS           60  60 100  87
ROUSSEAU        JEAN-JACQUES    80  80  93  55
CORNEILLE       PIERRE         100  20  56  75
```

Rappelons que la primitive ⍪ est nécessaire sinon on obtient le message "DOMAIN ERROR" qui signifie l'impossibilité d'associer des données de type numérique (les NOTES) et des données de type caractère (les NOMS).
Notons aussi qu'il n'est pas nécessaire, comme dans le cas d'un vecteur de notes, de restructurer ces notes avec la primitive "⍴" puisqu'elles sont déjà sous la forme d'un tableau de 12 lignes.

Pour ajouter la moyenne des 4 notes dans une 5 ième colonne:

```
      +/NOTES          fait la somme ligne par ligne
298 288 328 271 274 335 208 284 382 307 308 251
```

Pour avoir la moyenne il suffit de diviser par 4:

```
      (+/NOTES)÷4
74.5 72 82 67.75 68.5 83.75 52 71 95.5 76.75 77 62.75
```

Soyons généreux en arrondissant la moyenne à l'entier supérieur:

```
      ⌈(+/NOTES)÷4      la primitive ⌈ fait l'arrondi à l'entier
75 72 82 68 69 84 52 71 96 77 77 63
```

Ajoutons à nouveau les NOMS:

```
      NOMS,⍪NOTES,⌈(+/NOTES)÷4
RIMBAULT        ARTHUR          80  78  45  95  75
DAUDET          ALPHONSE        97  85  46  60  72
ZOLA            EMILE           65 100  88  75  82
POQUELIN        JEAN-BAPTISTE   66  60  70  75  68
DESCARTES       RENE            78   0 100  96  69
LA FONTAINE     JEAN            95  80  80  80  84
BOILEAU         NICOLAS         72  55  48  33  52
LA BRUYERE      JEAN            50  60  98  76  71
PROUST          MARCEL          89 100  98  95  96
DIDEROT         DENIS           60  60 100  87  77
ROUSSEAU        JEAN-JACQUES    80  80  93  55  77
CORNEILLE       PIERRE         100  20  56  75  63
```

POUR PRESENTER LES RESULTATS PAR ORDRE DE MERITE:

On fait le tri numérique sur la moyenne comme suit:

```
      MOY←⌈(+/NOTES)÷4    moyenne arrondie à l'entier supérieur
      I←⍒MOY
      I    donne les indices des moyennes en ordre décroissant
9 6 3 10 11 1 2 8 5 4 12 7
```

il suffit ensuite de donner ces indices I pour afficher les résultats NOMS, NOTES et MOY dans le bon ordre.

```
      NOMS[I;],⍕NOTES[I;],MOY[I]
PROUST          MARCEL          89 100  98  95  96
LA FONTAINE     JEAN            95  80  80  80  84
ZOLA            EMILE           65 100  88  75  82
DIDEROT         DENIS           60  60 100  87  77
ROUSSEAU        JEAN-JACQUES    80  80  93  55  77
RIMBAULT        ARTHUR          80  78  45  95  75
DAUDET          ALPHONSE        97  85  46  60  72
LA BRUYERE      JEAN            50  60  98  76  71
DESCARTES       RENE            78   0 100  96  69
POQUELIN        JEAN-BAPTISTE   66  60  70  75  68
CORNEILLE       PIERRE         100  20  56  75  63
BOILEAU         NICOLAS         72  55  48  33  52
```

POUR PRESENTER LES RESULTATS PAR ORDRE ALPHABETIQUE:

Mettons l'alphabet dans le vecteur ALPHA:

```
      ALPHA←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

La primitive ⍋ fait le tri des lignes selon l'alphabet ALPHA:

```
      I←ALPHA ⍋ NOMS
      I
7 12 2 5 10 8 6 4 9 1 11 3
```

Comme dans le cas d'un tri numérique, le résultat est un vecteur d'indices. l'indice 1 correspond au premier par ordre alphabétique, l'indice 2 est le 2ième, etc...

La valeur de I donne l'ordre (indices) pour les matrices NOMS et NOTES et pour le vecteur MOY.

```
      NOMS[I;],⍕NOTES[I;],MOY[I]
BOILEAU         NICOLAS         72  55  48  33  52
CORNEILLE       PIERRE         100  20  56  75  63
DAUDET          ALPHONSE        97  85  46  60  72
DESCARTES       RENE            78   0 100  96  69
DIDEROT         DENIS           60  60 100  87  77
LA BRUYERE      JEAN            50  60  98  76  71
LA FONTAINE     JEAN            95  80  80  80  84
POQUELIN        JEAN-BAPTISTE   66  60  70  75  68
PROUST          MARCEL          89 100  98  95  96
RIMBAULT        ARTHUR          80  78  45  95  75
ROUSSEAU        JEAN-JACQUES    80  80  93  55  77
ZOLA            EMILE           65 100  88  75  82
```

Ce dernier exemple illustre bien la puissance des primitives du langage APL...

2.5-MANIPULONS UN PEU DE TEXTE:

Soit le texte suivant:

```
     TEXTE←'Maître corbeau sur un arbre perché tenait dans son bec un fromage'
```

Pour compter les mots du texte (le séparateur de mots étant l'espace (caractère ' '). Il suffit donc de compter le nombre de caractères "espace" et d'ajouter 1.

```
     TEXTE=' '   chaque caractère ' ' est indiqué par un 1
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0
     +/TEXTE=' '   on fait la somme des 1
11
               il y a 11 espaces
     1++/TEXTE=' '
12
               il y a donc 12 mots
```

Combien de lettres 'a' y a-t-il dans ce texte?

```
     +/TEXTE='a'
6
```

Combien y a-t-il de voyelles 'aeiou' ?

```
     TEXTE∈'aeiou'
0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1
 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 1
               la primitive ∈ indique par un 1 chaque voyelle du
               texte.
     +/TEXTE∈'aeiou'
21
               si l'on veut tenir compte des majuscules il suffit
               de les ajouter entre les apostrophes 'aeiuoAEIOU'.
```

Pour extraire le premier mot du texte on cherche la position du premier caractère ' ' et on prend ce nombre de caractères:

```
     TEXTE⍳' '
7
               la primitive ⍳ appelée iota "cherche" ici la position
               du premier caractère ' ' dans la variable caractère
               TEXTE. Notez que la primitive ⍳ n'a pas ici le même
               sens que dans l'exécution de ⍳N. Nous y reviendrons.

     7↑TEXTE
Maître
               la primitive ↑ extrait le nombre de caractères
               trouvés par le ⍳.
     (TEXTE⍳' ')↑TEXTE
Maître
```

Enfin, pour chercher un mot dans un texte, nous disposons d'une fonction spéciale (il y en a beaucoup d'autres dans cette version d'APL) qui s'écrit DSS (SS pour Search String).

```
     TEXTE DSS 'arbre'
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

le résultat est un vecteur binaire, et l'on a un 1 à la position où le mot a été trouvé.

```
     TEXTE DSS 'un'
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
```

on a ici 2 occurences du mot cherché.

On peut donc savoir si un mot existe dans un texte, par:

```
     ∨/TEXTE DSS 'arbre'
1
```

∨/ donne un 1 si on a au moins un 1 dans le vecteur binaire. La primitive ∨ est un <u>ou logique</u>

```
     ∨/TEXTE DSS 'maison' le mot "maison" n'est pas dans le texte.
0
```

On peut compter le nombre de mots, avec:

```
     +/TEXTE DSS 'un'
2
```

Enfin on peut connaitre la position d'un mot dans le texte par:

```
     (TEXTE DSS 'arbre')/⍳⍴TEXTE
23
```

Décomposons cette dernière expression APL à partir (bien sûr!) de la droite:

```
     ⍴TEXTE
65
```

le nombre de caractères du texte

```
     ⍳⍴TEXTE
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 3
0 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
 57 58 59 60 61 62 63 64 65
```

les indices de chaque caractère du texte

enfin, la primitive / sélectionne les indices sur le vecteur binaire produit par (TEXTE DSS 'arbre').

Nous avons déjà rencontré plusieurs primitives à travers des exemples en **mode d'exécution immédiate** c'est à dire en tapant les "expressions APL" sans écrire de "programme". Les fonctions primitives d'APL sont nombreuses et elles obéissent à des règles précises qui seront données dans l'annexe A. Les lecteurs soucieux de "règles" et de "classification" des primitives peuvent déjà s'y reporter. Nous continuerons à découvrir les primitives avec des exemples concrets et à travers la programmation.

CHAPITRE 3

LA PROGRAMMATION AVEC APL

APL permet d'exécuter, comme on vient de le voir, des expressions directement tapées au clavier, un peu comme une calculatrice. Mais, pour résoudre des problèmes plus complexes et pour suivre des algorithmes impliquant des choix, il faut pouvoir conserver ces **expressions APL** dans des PROGRAMMES.

Avec APL, au lieu d'utiliser le mot "**programme**" on dit "**fonction définie**" ou plus simplement "**fonction**".

On a donc:
- les **fonctions primitives** ou **primitives** qui constituent le langage de base.
- et les **fonctions définies** ou **fonctions** qui sont crées par le programmeur et qui sont donc des extensions du langage de base.

Pour définir une **fonction** nous avons le choix entre deux éditeurs:
- l'éditeur standard commun à tous les systèmes APL.
- l'éditeur "**plein écran**" qui est propre à la version APL*PLUS/PC que nous utilisons.

Utilisons dans un premier temps l'éditeur standard qui sera suffisant pour définir les premières fonctions élémentaires que nous écrirons au début. Mais nous suggèrerons l'utilisation de l'éditeur **plein écran** dès qu'il s'agira de fonctions plus longues et pour la correction de fonctions déjà définies.

3.1 - PREMIERES FONCTIONS

- Fonction MOYENNE

Nous avons vu au paragraphe 2.4 l'expression permettant de calculer la moyenne d'une série de notes. Pour éviter de taper cette expression à chaque fois que l'on veut faire une moyenne on va écrire une **fonction** appelée **MOYENNE**.

Avec l'**éditeur standard**, pour définir la fonction **MOYENNE** on tapera d'abord le symbole ∇ qui indique l'entrée en mode de définition suivi de la **ligne de définition** de la fonction.
Par exemple:

```
      ∇ M←MOYENNE N      (suivi de ENTER)
[1]
```

la ligne de définition comprend ici:
- le nom de la fonction: MOYENNE
- le nom de l'**argument**: N (valeur donnée en entrée)
- le nom du **résultat**: M (valeur produite à la sortie)

APL numérote les lignes entre [ ],
on entre chaque ligne suivie de ENTER,
pour terminer on tape à nouveau le ∇.

cela donne au complet ce qui suit:

```
     ∇ M←MOYENNE N
[1] M←(+/N)÷⍴N
[2] ∇
```

La fonction est définie. Essayons-là:

```
      MOYENNE 3 4 5 6
4.5
      SALAIRES←28000 22500 29000 33000 31450 31200 33200
      MOYENNE SALAIRES
29764.28571
```

On remarque que le nom N choisi pour l'**argument** dans la fonction est totalement indépendant des noms de variables (par exemple SALAIRES) utilisés au moment de l'exécution. La fonction MOYENNE prend la valeur contenue dans la variable SALAIRES , met cette valeur dans la variable **N** pour faire ses propres calculs. N est une variable "interne" à la fonction MOYENNE.

A quoi sert la variable M? M est aussi une variable "interne" à la fonction dont la valeur à la fin de l'exécution est rendue comme résultat de l'exécution. Ce résultat pouvant être affecté dans une variable. On dit que la fonction produit un **résultat explicite** parce que ce résultat peut être "transmis". On peut donc faire:

```
      SALMOYEN←MOYENNE SALAIRES
```
et le résultat de l'exécution de la fonction MOYENNE est affecté dans la variable SALMOYEN.
```
      SALMOYEN
29764.28571
```

Notez, car c'est important, que les variables N et M disparaissent après l'exécution de la fonction. Elles n'ont existé que pendant l'exécution. En effet:
```
      N
VALUE ERROR
      N
      ^
```

Plus, si la variable N vaut par exemple 1984 avant l'exécution de la fonction MOYENNE, cette valeur (1984) est "cachée" pendant l'exécution et elle est restaurée après. Exemple:
```
      N←1984
      MOYENNE 2 4 6 8 10
6
      N
1984
```

Le rôle des arguments et du résultat explicite est extrêmement important et il permet une grande souplesse en programmation. Ce rôle pourra être parfaitement assimilé avec l'expérience sans qu'il soit nécessaire d'y ajouter plus de théorie!

Revenons à notre fonction MOYENNE. Quelle serait la moyenne des salaires après une augmentation placée dans la variable AUGMENTATION:
```
      AUGMENTATION←560 500 600 800 750 900 1000
      MOYENNE (SALAIRES + AUGMENTATION)
30494.28571
```
rappelons que le résultat s'affiche car il n'y a pas d'affectation dans une variable.

La fonction MOYENNE peut maintenant être utilisée comme une fonction primitive d'APL, elle remplace toute une expression APL. Voyons un exemple d'utilisation.

- Quelques statistiques

Utilisons la fonction MOYENNE dans une fonction appelée STATS qui imprime quelques caractéristiques statistiques sur les salaires précédents:
```
   ∇STATS SAL
[1] 'LE SALAIRE LE PLUS ELEVE EST: ',⍕ ⌈/SAL
[2] 'LE SALAIRE LE MOINS ELEVE EST:',⍕ ⌊/SAL
[3] 'LA MOYENNE DES SALAIRES EST:  ',⍕ MOYENNE SAL
[4] 'LE TOTAL DES SALAIRES EST:    ',⍕ +/SAL
[5] ∇
```

Exécutons la fonction STATS avec comme argument le vecteur SALAIRES précédent:

```
      STATS SALAIRES
LE SALAIRE LE PLUS ELEVE EST: 33200
LE SALAIRE LE MOINS ELEVE EST:22500
LA MOYENNE DES SALAIRES EST:  29764.28571
LE TOTAL DES SALAIRES EST:    208350
```

Remarque: Contrairement à la fonction MOYENNE, la fonction STATS ne produit pas de résultat explicite elle imprime un résultat qui est considéré comme un résultat final n'ayant pas lieu d'être affecté dans une variable pour un usage ultérieur.

Si vous désirez modifier la fonction STATS nous vous suggérons d'utiliser l'éditeur "plein écran".

EXERCICE (pour ceux qui aiment les statistiques !)
Ecrire une fonction qui calcule l'**écart-type** d'une série statistique.

- **trouver les diviseurs d'un nombre**

La fonction primitive | appelée **modulo** permet de trouver le reste de la division d'un nombre par un autre. Par exemple le reste de la division de 20 par 3, puis de 20 par 4, est obtenu par:

```
      3|20        en effet  20=(3×6)+2
2
      4|20        reste de la division de 20 par 4
0
```

4 est diviseur de 20 puisque le reste est 0

On peut obtenir directement le reste de la division de 20 par chacun des entiers de 1 à 20 obtenus par ⍳20.

```
      (⍳20)|20
0 0 2 0 0 2 6 4 2 0 9 8 7 6 5 4 3 2 1 0
```

Quand le reste de la division est 0 c'est un diviseur.

```
      0=(⍳20)|20
1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
```

En utilisant ensuite la primitive / appliquée sur tous les entiers de 1 à 20, on arrive au résultat cherché.

```
      V←0=(⍳20)|20
      V/⍳20
1 2 4 5 10 20
```

Ecrivons la fonction DIVISEURS:

```
    ∇D←DIVISEURS N;V
[1] V←0=(⍳N)|N
[2] D←V/⍳N
[3]∇
```

V est une variable "interne" à la fonction DIVISEURS en la plaçant dans la ligne de définition séparée par un ; elle est dite **variable locale** c'est à dire une variable qui n'existe que pendant l'exécution de la fonction DIVISEURS.

Exécution:

```
      DIVISEURS 20
1 2 4 5 10 20
      DIVISEURS 100
1 2 4 5 10 20 25 50 100
      DIVISEURS 1000
1 2 4 5 8 10 20 25 40 50 100 125 200 250 500 1000
```

Vérifions que la variable V n'existe plus:

```
      V
VALUE ERROR
      V
      ∧
```

Un nombre est premier s'il n'est divisible que par 1 et par lui-même, autrement dit s'il n'a que 2 diviseurs:

```
      DIVISEURS 147
1 3 7 21 49 147
      ⍴DIVISEURS 147
6
      2=⍴DIVISEURS 147    147 n'est pas premier
0

      2=⍴DIVISEURS 151    151 est premier
1
```

Cette méthode n'est évidemment pas la plus efficace pour trouver si un nombre est premier ou non. Mais c'est un moyen d'exploration pédagogique.

- <u>Premier mot d'un texte</u>

Soit le texte déjà utilisé au chapitre 2:

```
      TEXTE
Maître corbeau sur un arbre perché tenait dans son bec un fromage
```

On peut trouver la position du premier caractère blanc par:

```
      TEXTE⍳' '
7
```

On dispose ensuite d'une primitive AFL (↑), appelée "take" en anglais, qui permet d'**extraire** le nombre de caractères voulus:

```
      7↑TEXTE
Maître
      (TEXTE≠' ')↑TEXTE
Maître
```

On écrit donc la fonction PMOT (pour Premier MOT):

```
  ∇ MOT←PMOT T
[1] MOT←(T≠' ')↑T
[2]∇
```

Notez que le MOT est suivi d'un caractère blanc à la fin. On pourrait l'enlever par exemple avec MOT←((T≠' ')-1)↑T ou MOT←(¯1+T≠' ')↑T pour éliminer une parenthèse.

Exécution:

```
      PMOT TEXTE
Maître
      PMOT 'Au clair de la lune'
Au
```

EXERCICE: Extraire la première phrase d'un texte sachant que chaque phrase est terminée par un point(.). Que faire si les phrases du texte peuvent aussi se terminer par un point d'exclamation (!) ou un point d'interrogation (?) ?

- **Enlever le premier mot d'un texte**

La primitive (↓) permet d'enlever des caractères (c'est l'inverse de ↑).

```
      7↓TEXTE
corbeau sur un arbre perché tenait dans son bec un fromage
```

On peut donc faire:

```
      (TEXTE≠' ')↓TEXTE
corbeau sur un arbre perché tenait dans son bec un fromage
```

D'où la fonction SPMOT (Sauf Premier MOT):

```
  ∇TT←SPMOT T
[1] TT←(T≠' ')↓T
[2] ∇
```

Exécution:

```
      SPMOT TEXTE
corbeau sur un arbre perché tenait dans son bec un fromage
```

Enchaînons les exécutions pour avoir les 2 et 3ième mot du texte:

```
      PMOT SPMOT TEXTE
corbeau
      PMOT SPMOT SPMOT TEXTE
sur
```

Les fonctions que nous venons de voir ont toutes un **argument** (à droite), elles sont dites **monadiques**. Il est possible d'écrire des fonctions avec 2 arguments dites **dyadiques**. C'est ce que nous allons voir.

3.2 - FONCTIONS AVEC 2 ARGUMENTS

- Recherche d'un mot dans un texte

Nous avons vu au paragraphe 2.5 une expression qui permet de trouver la position d'un mot dans un texte. Ecrivons une fonction dont le nom sera CHERCHE avec comme argument de gauche le texte et comme argument de droite le mot cherché.

```
    ∇ POS←TXT CHERCHE MOT
[1] POS←(TXT DSS MOT)/⍳⍴TXT
[2] ∇
```

Rappelons que le choix des noms de variable est arbitraire. Nous choisissons donc des noms assez significatifs: TXT pour le texte, MOT pour le mot et POS pour la position.

```
      TEXTE
Maitre corbeau sur un arbre perché tenait dans son bec un fromage
      TEXTE CHERCHE 'arbre'
23
```

le mot "arbre" débute au 23 ième caractère.

```
      TEXTE CHERCHE 'un'
20 56
```

on a 2 occurences du mot "un"

```
      TEXTE CHERCHE 'saison'
```

la réponse est vide: le mot n'est pas dans le texte

Pour savoir combien de fois un mot est répété dans un texte on peut (ce n'est pas la seule façon) appliquer la primitive ⍴ qui compte le nombre d'éléments:

```
      TEXTE CHERCHE 'un'
20 56
      ⍴TEXTE CHERCHE 'un'
2
      ⍴TEXTE CHERCHE 'arbre'
1
      ⍴TEXTE CHERCHE 'saison'
0
```

- intersection d'ensembles

Nous avons aussi rencontré au paragraphe 2.2 une expression qui permettait de trouver les éléments communs de 2 ensembles de nombres. Définissons une fonction INTER (pour intersection) avec 2 arguments (A argument de gauche et B argument de droite):

```
    ∇R←A INTER B
[1] R←(A∈B)/A
[2] ∇
```

on a encore un **résultat explicite** (la variable R) pour transmettre le résultat de l'exécution.

```
      12 4 9 8 5 7 10 INTER  7 8 10 11 13 14 16 18 19
8 7 10
```

on peut aussi enchainer l'exécution pour l'intersection de plusieurs vecteurs:

```
      A←12 4 9 8 5 7 10
      B←7 8 10 11 13 14 16 18 19
      C←2 4 6 8 10 12 14 16 18 20 22 24
      B INTER C
8 10 14 16 18
      A INTER B INTER C
8 10
```

B INTER C qui s'exécute en premier transmet son son résultat qui devient argument de droite pour la seconde exécution de la fonction INTER. C'est exactement la même principe que dans 3+4+5 : le résultat de 4+5 est l'argument pour la 2ième addition.

Soit un nouvel ensemble D dont l'intersection avec l'ensemble C est vide. Par exemple:

```
      D←5 9 13 17
      C INTER D

Aucun résultat ne s'affiche. Vérifions le nombre d'éléments avec
la primitive ⍴:
      ⍴C INTER D   on a bien le vide (0 élément)
0
```

Application

Nous avons vu au paragraphe 3.1 la fonction DIVISEURS. Nous pouvons maintenant trouver les diviseurs communs de 2 nombres. Par exemple:

```
      DIVISEURS 72
1 2 3 4 6 8 9 12 18 24 36 72
      DIVISEURS 90
1 2 3 5 6 9 10 15 18 30 45 90
      (DIVISEURS 72) INTER (DIVISEURS 90)
1 2 3 6 9 18
```

et pour trouver le PGCD (Plus Grand Commun Diviseur) on peut faire:

```
      ⌈/(DIVISEURS 72) INTER (DIVISEURS 90)
18
```

notez que la recherche du plus grand diviseur commun se fait de façon plus efficace avec l'algorithme d'Euclide. La présente méthode exige beaucoup de mémoire avec des grands nombres. Elle n'a qu'un intéret théorique.

3.3 - FONCTION AVEC BRANCHEMENTS

Tous les problèmes ne peuvent évidemment pas être résolus par une seule ligne de code APL! Et bien qu'APL réduise considérablement le nombre de boucles et de branchements comparativement aux autres langages, des boucles et des branchements s'avèrent nécessaires dès que les algorithmes deviennent plus complexes. Nous choisirons d'aborder les branchements avec le problème très concret du **remboursement d'un emprunt.**

problème
On veut imprimer le calendrier de remboursement d'un emprunt à mensualités constantes, connaissant le montant du prêt, le taux annuel d'intérêt et le versement mensuel fixe.
Soit, par exemple, les données suivantes:

```
PRET←5000      prêt de 5000 $
TAUX←12        taux annuel de 12%
VERSE←150      versement mensuel de 150 $
```

Donc, si vous avez fait cet emprunt de 5000$ vous aurez dès le premier mois de l'intérêt à payer. Cet intérêt est calculé comme suit: un taux de 12% cela veut dire 0,12$ pour chaque dollar annuellement. Il faut encore diviser par 12 pour obtenir l'intérêt pour un mois, ainsi:

```
      PRET : TAUX÷100     donne l'intérêt annuel
600
      PRET : TAUX÷1200    donne l'intérêt mensuel cherché
50
```

Conservons cette valeur dans la variable INTERET:

```
      INTERET←PRET:TAUX÷1200
      INTERET
50
```

Le versement (VERSE) étant de 150$ et l'intérêt calculé étant de 50$, le remboursement de capital est obtenu en déduisant du versement fixe (VERSE) la valeur de l'intérêt. On conservera cette valeur dans la variable REMBOURSE par:

```
      REMBOURSE←VERSE-INTERET
      REMBOURSE
100
```

Enfin puisque nous remboursons 100$ de capital la valeur résiduelle du prêt (ou solde du capital) est la valeur du prêt moins le remboursement de capital, c'est-à-dire:

```
      PRET←PRET-REMBOURSE  on déduit du PRET la valeur de REMBOURSE

      PRET
4900
```

Pour le premier versement (150$), l'intérêt est de 50$.
le remboursement de capital est de 100$, et solde du prêt est
de 4950$. On peut répéter les mêmes instructions pour le 2ième
versement, le montant du PRET étant maintenant de 4950$:

```
      INTERET←PRET×TAUX÷1200
      INTERET
49
```

l'intérêt est cette fois calculé sur le solde du prêt
de 4950$

```
      REMBOURSE←VERSE-INTERET
      REMBOURSE
101
      PRET←PRET-REMBOURSE
      PRET
4799
```

On voit qu'à chaque versement la part d'intérêt diminue puisque
le PRET diminue, et que la part de remboursement augmente. Mais
devra-t-on répéter ainsi chacune des opérations précédentes pour
chaque versement? (c'est ce qu'il faudrait faire avec une
calculatrice non programmable!) Non, nous avons la possibilité
de faire répéter par l'ordinateur les mêmes instructions dans
une boucle. Ecrivons d'abord une fonction que nous appelerons
EMPRUNT qui exécute les instructions précédentes et pour cela
utilisons l'**éditeur plein écran** propre à APL*PLUS/PC, comme
suit:

```
      )EDIT EMPRUNT
```

On passe ici sous le controle de l'éditeur plein écran (voir
le résumé des commandes en annexe B). L'éditeur considère
EMPRUNT comme une variable de type caractère. Pour en faire
une **fonction** on tape:

CTRL A, puis on répond à la ligne 25 par F pour FONCTION

on peut alors taper le contenu de la fonction EMPRUNT, comme suit:

```
[0]  EMPRUNT
[1]  INTERET←PRET×TAUX÷1200
[2]  REMBOURSE←VERSE-INTERET
[3]  PRET←PRET-REMBOURSE
[4]  INTERET,REMBOURSE,PRET
```

Pour sortir de l'éditeur on fait: CTRL E et on retrouve le
texte initial (avant l'appel de l'éditeur). Il facile de
reconnaitre:

- la ligne [1] qui calcule l'intérêt
- la ligne [2] qui calcule le remboursement
- la ligne [3] qui déduit du PRET la valeur du remboursement
- la ligne [4] qui met consécutivement (concatène) les valeurs des variables INTERET, REMBOURSE et PRET, et qui les imprime puisqu'il n'y a pas d'affectation dans une variable.

Demandons l'exécution de la fonction EMPRUNT après avoir rumis dans la variable PRET la valeur de 5000$.

```
      PRET←5000
      EMPRUNT
50 100 4900
```

Nous retrouvons le premier versement

```
      EMPRUNT   2ième versement
49 101 4799
      EMPRUNT   3ième versement
47.99 102.01 4696.99
```

Nous n'allons pas taper le mot EMPRUNT pour chaque versement! Nous allons faire un branchement à la ligne [5] comme suit:

```
[5] →1        expression qui signifie "retourne à la ligne 1".
```

En utilisant l'éditeur pour corriger la fonction EMPRUNT, cela donne ce qui suit:

```
[0]  EMPRUNT
[1]  INTERET←PRET×TAUX÷1200
[2]  REMBOURSE←VERSE-INTERET
[3]  PRET←PRET-REMBOURSE
[4]  INTERET,REMBOURSE,PRET
[5]  →1
```

Prenons cette fois, pour limiter la longueur du résultat, un PRET de 2000$.

```
      PRET←2000
      EMPRUNT
20 130 1870
18.7 131.3 1738.7
17.387 132.613 1606.087
16.06087 133.93913 1472.14787
14.7214787 135.2785213 1336.869349
13.36869349 136.6313065 1200.238042
12.00238042 137.9976196 1062.240423
10.62240423 139.3775958 922.8628268
9.228628268 140.7713717 782.0914551
7.820914551 142.1790854 639.9123697
6.399123697 143.6008763 496.3114934
4.963114934 145.0368851 351.2746083
3.512746083 146.4872539 204.7873544
2.047873544 147.9521265 56.83522791
0.5683522791 149.4316477 ¯92.59641981
¯0.9259641981 150.9259642 ¯243.522384
¯2.43522384 152.4352238 ¯395.9576078
```

etc ...

La fonction EMPRUNT ne s'arrête pas d'elle-même il faut interrompre l'exécution par CTRL ESC .

On voit que le PRET devient négatif... Il faut donc améliorer le contrôle du branchement en faisant un "branchement conditionnel". On branchera à la ligne [1] tant que le PRET sera positif. Cela se fait en corrigeant la ligne [5] comme suit:

```
[0]  EMPRUNT
[1]  INTERET←PRET×TAUX÷1200
[2]  REMBOURSE←VERSE-INTERET
[3]  PRET←PRET-REMBOURSE
[4]  INTERET,REMBOURSE,PRET
[5]  →(PRET>0)/1
```

La ligne [5] signifie "branche à 1 tant que PRET>0".

```
      PRET←2000
      EMPRUNT
20 130 1870
18.7 131.3 1738.7
17.387 132.613 1606.087
16.06087 133.93913 1472.14787
14.7214787 135.2785213 1336.869349
13.36869349 136.6313065 1200.238042
12.00238042 137.9976196 1062.240423
10.62240423 139.3775958 922.8628268
9.228628268 140.7713717 782.0914551
7.820914551 142.1790854 639.9123697
6.399123697 143.6008763 496.3114934
4.963114934 145.0368851 351.2746083
3.512746083 146.4872539 204.7873544
2.047873544 147.9521265 56.83522791
0.5683522791 149.4316477 ¯92.59641981
```

L'exécution s'arrête maintenant lorsque la valeur du PRET est négative. Cependant le résultat n'est pas lisible à cause des nombres décimaux et le manque d'alignement. Nous pouvons améliorer ce résultat en modifiant la ligne [4] comme suit:

```
[4]   9 2▼INTERET,REMBOURSE,PRET
```

```
[0]  EMPRUNT
[1]  INTERET←PRET×TAUX÷1200
[2]  REMBOURSE←VERSE-INTERET
[3]  PRET←PRET-REMBOURSE
[4]  9 2▼INTERET,REMBOURSE,PRET
[5]  →(PRET>0)/1
```

```
      PRET←2000
      EMPRUNT
    20.00    130.00   1870.00
    18.70    131.30   1738.70
    17.39    132.61   1606.09
    16.06    133.94   1472.15
    14.72    135.28   1336.87
    13.37    136.63   1200.24
    12.00    138.00   1062.24
    10.62    139.38    922.86
     9.23    140.77    782.09
     7.82    142.18    639.91
     6.40    143.60    496.31
     4.96    145.04    351.27
     3.51    146.49    204.79
     2.05    147.95     56.84
      .57    149.43    ¯92.60
```

Il y aurait plusieurs améliorations à apporter à la fonction EMPRUNT:

1- faire un branchement à un numéro de ligne n'est pas recommandé car, en cas de modification de la fonction (insersion ou destruction de lignes), les branchements sont à modifier. On recommande l'utilisation d'étiquettes. On dira par exemple:

```
[1] ENCORE: INTERET←PRET×TAUX÷1200
 .....
[5]   →(PRET>0)/ENCORE
```

Le mot ENCORE est appelé une **étiquette.**

2- les branchements peuvent être rendus plus "lisibles" en utilisant une fonction "utilitaire" très simple, la fonction SI de la façon suivante:

```
[5]  →ENCORE SI PRET>0
```

la fonction SI est définie (une fois pour toutes) par:

```
    ∇ R← E SI C
[1]   R←C/E
      ∇
```

3- il faudrait "ajuster" le dernier paiement pour éviter de payer en trop!

4- On pourrait numéroter les paiements et faire le total des intérêts payés... etc ...

3.4 - COMMANDES DE SYSTEME ELEMENTAIRES

Toutes les **commandes de système** en APL commencent par une parenthèse droite ")", et doivent être exécutées seules sur une ligne. Voici les commandes essentielles pour débuter:

)LIB permet de connaître le nom des **espaces de travail** ("workspaces") et des **fichiers** sur disquette.

On peut faire)LIB 0 pour la disquette de gauche
)LIB 1 pour la disquette de droite

Les espaces de travail ont l'extension **.AWS**
les fichiers ont l'extension **.ASF**

)LOAD n nom
cette commande amène (ou charge) en mémoire l'**espace de travail "nom"** qui se trouve sur la disquette "n" (n= 0 ou 1).
Exemple:)LOAD 1 TORTUE

Si l'espace de travail est sur l'unité de disquette 0 on peut omettre le 0. Exemple:)LOAD TORTUE

)FNS pour connaître les noms des fonctions de l'espace de travail qui se trouve en mémoire.

)VARS pour connaître les noms des variables de l'espace de travail qui se trouve en mémoire.

)ERASE liste
pour effacer une **liste** d'objets APL (variables ou fonctions de l'espace de travail qui se trouve en mémoire. Ex:)ERASE A TEXTE STATS

)SAVE pour conserver sur disquette l'espace de travail qui est en mémoire. Si l'espace de travail est nouveau (il vient d'être créé en mémoire), alors il faut lui donner un nom, par exemple ESSAI, et on fait:
)SAVE ESSAI

)WSID permet de connaître le nom de l'espace de travail qui est en mémoire.

)WSID nom
permet de donner un "nom" à l'espace de travail qui est en mémoire.

ANNEXE

L'EDITEUR PLEIN-ECRAN D'APL * PLUS / PC

* * L'éditeur plein écran d' APL * PLUS / PC * *

La commande **)EDIT "nom"** permet d'éditer l objet APL **"nom"** qui peut être soit un **vecteur caractère**, une **matrice caractère**, ou **une fonction APL**.

Pour éditer un objet dont le nom est ABC on fait:

)EDIT ABC

- **Si ABC n'existe pas** dans l'espace de travail, l'éditeur le crée, affiche la ligne [0] , et l'objet ABC est considéré comme un **vecteur caractère**.
  (voir la ligne 25 de l écran)

- **Si ABC existe déjà**, sa valeur s'affiche à l'écran avec son type (vecteur, matrice, fonction), indiqué à la ligne 25 de l'écran.

On peut alors **ajouter** du texte, déplacer le curseur, insérer ou effacer des lignes, etc... (Voir les commandes ci-dessous).

NB: Les commandes de mouvement du curseur ainsi que les commandes d'insertion et de suppression de lignes sont valables aussi en dehors de l'éditeur. Elles sont données dans le résumé ci-après.

- **Pour changer le type d'un objet** (vecteur, matrice, fonction):

Ctrl A répondre F, V ou M à la ligne 25 de l'écran.

Pour créer une fonction, par exemple la fonction TRUC (on suppose qu'il n'y a pas de variable de même nom dans l'espace de travail), on fait:

)EDIT TRUC

On obtient la ligne [0] et TRUC est considéré comme une variable (voir la ligne 25 de l'écran).
On fait alors CTRL A et on répond par F pour changer TRUC en fonction APL.
Puis on écrit la ligne[0] (ligne de définition) de TRUC, ses arguments et ses variables locales s'il y a lieu.
On écrit ensuite ligne par ligne la fonction TRUC.

- **Pour sortir de l'éditeur:**

a) **Ctrl E** (E=Exit)

L'objet APL est installé dans l'espace de travail selon le type qu'il a reçu et il remplace l'ancienne version .

Attention:
Si on a changé le type variable pour le type fonction avec Ctrl A , la fonction ne peut être installée s il y a déjà un autre objet APL de même nom. C'est une protection pour éviter de détruire une fonction ou une variable de même nom dans l'espace de travail. On obtient le message DEFN ERROR (erreur de définition).

b) **Ctrl Q** (Q=Quit)

On quitte sans créer l'objet ABC ou sans modifier l'ancienne version. C'est comme si on n'avait rien fait!

- **Pour déplacer un bloc de texte de une ou plusieurs lignes:**

On marque la ligne de début (où se trouve le curseur) avec: **Ctrl T** (T=Tag)
On marque la ligne de fin (en déplaçant le curseur) avec: **Ctrl T**

On déplace le curseur à la ligne où on veut insérer le bloc de lignes et on fait : **Ctrl M** (M=Move)

Le bloc de texte est alors déplacé et les lignes sont renumérotées.

- **Pour duplicater** (recopier) **un bloc de texte:**

On marque le bloc comme précédemment (Ctrl T), puis:

Ctrl C (C=Copy)

Cela copie le bloc de texte sans effacer l'original, à la position du curseur. Le bloc reste marqué et on peut répéter Ctrl C

- **Pour détruire un bloc:**

Un bloc marqué par Ctrl T peut être détruit par:

Ctrl D (D=Delete)

et les lignes sont renumérotées.

- **Pour couper une ligne en deux:**

Ctrl B (B=Break)

Cette commande coupe la ligne à l'endroit du curseur.

- **Pour assembler 2 lignes:**

Ctrl G (G=Glue)

La ligne où se trouve le curseur et la suivante sont rassemblées, mises bout à bout.

- **Pour insérer un objet extérieur :**

Ctrl F (F=Fetch)

Il faut répondre à la ligne 25 de l'écran le nom de l'objet (variable caractère ou fonction) à insérer après la position du curseur.

- **Pour sortir à l'extérieur une partie du texte:**
Le texte doit être "marqué" avec CTRL T, puis on fait:

Ctrl V

Donner à la ligne 25 de l'écran le nom de la variable que l'on veut créer. Ce nom ne doit pas déjà exister dans votre espace de travail.

- **Pour remplacer une partie de texte** (un mot ou une chaine de caractère), on fait:

Ctrl R (R=Replace)

Donner à la ligne 25 de l'écran la chaine de caractère à remplacer, puis la chaine de caractère de remplacement. Enfin, répondre si l'on veut remplacer chacune des occurences ou remplacer dans tout le texte.

- **Pour localiser une partie du texte:**

Ctrl L (L=Locate)

Donner le texte à la ligne 25. Le curseur se positionne à la première occurence du texte.

Ctrl N (N=Next)

Cherche l occurence suivante dans le même texte.

- **Pour enlever ou remettre la numérotation des lignes:**

Ctrl O

R E S U M E

Alt F3 pour insérer une ligne avant celle du curseur

Alt F4 pour détruire la ligne

Alt → efface la fin de la ligne à partir du curseur

Alt ← efface le début de la ligne

Ctrl End déplace le curseur à la fin de la ligne

← (backspace) efface le caractère à gauche du curseur

Del efface le caractère à la position du curseur

Ctrl A pour changer le type

Ctrl B pour couper une ligne où se trouve le curseur

Ctrl C pour duplicater un bloc de texte

Ctrl D pour détruire un bloc de texte déjà marqué par Ctrl T

Ctrl E pour sortir du mode d'édition

Ctrl F pour insérer un bloc venant de l'extérieur

Ctrl G pour coller des lignes (mettre bout à bout)

Ctrl J pour remonter le curseur à la ligne demandée

Ctrl L pour localiser une partie de texte

Ctrl M pour déplacer un bloc "marqué" par Ctrl T

Ctrl O pour la numérotation des lignes (ON ou OFF)

Ctrl Q pour quitter, annuler l'édition

Ctrl R pour remplacer un mot ou une chaine de caractère

Ctrl T pour marquer un bloc de texte

Ctrl V pour créer un nouvel objet dans l'espace de travail